AN INCONSISTENCY SENSITIVE ARRANGEMENT ALGORITHM FOR ALGEBRAIC CURVES

Victor Milenkovic
Elisha Sacks

CSD TR #04-017
June 2004

An inconsistency sensitive arrangement algorithm for algebraic curves

Victor Milenkovic
University of Miami

Elisha Sacks
Purdue University

Abstract

We present a robust arrangement algorithm for algebraic curves based on floating point arithmetic. The algorithm generates a planar arrangement that is realizable by curves that are close to the input. We bound the running time and error in terms of the number of inconsistencies in the input. An inconsistency is a triple of monotonic curve segments that are assigned a cyclic vertical order over an open interval by the numerical algorithms. The number of vertices in the output is $V = 2n + N + \min(3kn, 0.5n^2)$ and the running time is $O(V \log n)$ for algebraic curves comprised of n monotonic segments with N intersection points and $k \in O(n^3)$ inconsistencies. The maximum distance between the realization curves and the input curves is $\epsilon + 3k\epsilon$ where ϵ is the curve intersection accuracy. We show experimentally that k is zero for generic inputs and is tiny for highly degenerate inputs. Hence, the algorithm running time on real-world inputs equals that of a standard sweep and the realization error equals the curve intersection error. The mean/max ϵ values are $10^{-12}/10^{-10}$ for curves of degree 40.

1 Introduction

We present a robust arrangement algorithm for algebraic curves based on floating point arithmetic. An algebraic curve is the zero set of a polynomial $F(x, y)$ with integer coefficients. A set of algebraic curves partitions the xy plane into an arrangement: a combinatorial structure comprised of cells, components of the plane away from curves; edges, smooth curve components away from other curves; and vertices, non-smooth curve points or curve intersection points. Computing arrangements is a core computational geometry task with diverse applications.

Arrangements can be computed by sweeping n curves with N intersections in $O((n+N)\log n)$ time. This runtime bound assigns unit cost to geometric operations, such as partitioning a curve into segments, intersecting two curves, and sorting vertices along an axis. The operations reduce to constructing algebraic numbers and computing the signs of polynomials at these numbers. The classical techniques for manipulating algebraic numbers incur a computational cost that grows rapidly with degree and bit complexity. The same problem arises in incremental insertion or in any other arrangement algorithm.

The challenge is to compute arrangements quickly despite their algebraic complexity. One research direction is to employ custom geometric algorithms, constructive root bounds, and floating point filters. This approach has led to practical arrangement algorithms for lines, circles, conics, and cubics. We present an alternate research direction that replaces algebraic computation with numerical computation.

We employ a sweep algorithm to compute the points where curve segments cross and the vertical y-order between crossings. The algorithm is purely combinatorial, except for calls to a numerical *crossing module* that computes the crossings of a pair of segments by the homotopy method: a proven algorithm that is orders of magnitude faster than algebraic computation and whose error is tiny. The only other numerical operation is comparison of floating point numbers, which is exact and has unit cost.

The drawback of the numerical approach is that even a tiny error in the crossing module output can lead to an inconsistency among the pairwise vertical segment orders. Inconsistency means that the pairwise orders are not extensible to a linear order, hence are not realizable by any input. The canonical example is three segments in cyclic order over an x-interval. When the standard sweep encounters inconsistencies, it can output an arrangement with a non-planar topology or a large numerical error. Our algorithm solves this problem by detecting and correcting inconsistencies. It outputs consistent vertical orders that are correct for a set of *realization curves* that are close to the input curves.

Differences between the computed and true vertical orders are inconsequential. Suppose the input models an object with accuracy α: the model is built from a prototype with α-accurate measurements or the object is manufactured from the model with an α-accurate process. If the realization curves are nearer than α to the input curves, our algorithm is indistinguishable from an exact algorithm because our input is indistinguishable from an input for which our output is exact. We show that the realization curves are nearer than 10^{-10} units to the input, which is smaller than any realistic α value.

Degenerate inputs are a concern in arrangement computation. Degeneracies occur when combinatorially distinct entities coincide geometrically, for example three curves meet at a point. They complicate the task of constructing a cell arrangement from the sweep output, which is straightforward for generic input. This paper describes our sweep algorithm for generic input; a companion paper describes an implemented arrangement algorithm that handles all degeneracies.

The advantage of the numerical approach is good asymptotic performance with small constant factors that are independent of degree and bit complexity. The number of vertices in the output is $V = 2n + N + \min(3kn, 0.5n^2)$ and the running time is $O(V \log n)$ for curves comprised of n monotonic segments with N intersection points and $k \in O(n^3)$ inconsistencies. The maximum distance between the realization curves and the input is $\epsilon + 3k\epsilon$ where ϵ is the curve intersection accuracy. We call the algorithm *inconsistency sensitive* because the running time and the accuracy are expressed in terms of k with optimal values at $k = 0$. We show experimentally that k is zero for generic inputs and is small for degenerate inputs. Hence, the running time on real-world inputs equals that of a standard sweep and the realization error equals the curve intersection error. The mean/max ϵ values are $10^{-12}/10^{-10}$ for curves of degree 40.

The rest of the paper is organized as follows. Section 2 surveys prior work on arrangement algorithms. Section 3 specifies the input to our sweep algorithm and sections 4 and 5 describe the algorithm. Section 6 explains how to convert algebraic curves into the algorithm input format. Section 7 presents empirical measurements of k and ϵ. Section 8 discusses our results.

2 Prior work

Halperin [6] surveys arrangements with a focus on linear objects. Yap [20] surveys robust computational geometry with a focus on exact methods.

Keyser et al [9] compute arrangements of non-degenerate rational parametric curves with an $O(n^2)$ algorithm. Arranging 12 curves of degree at most 4 with 80 bit coefficients takes 1142 seconds on a 400MHz Pentium 2. Halperin and Leiserowitz [7] compute arrangements of circles by a perturbation method.

LEDA [10] and CGAL [4] compute arrangements of line segments via generalizations of Bentley's sweep algorithm that employ filtered rational arithmetic. Wein [18] extends the CGAL arrangement algorithm to conics. Arranging 20 random conics takes 2 seconds on a 450MHz Pentium 2. Berberich et al [2] extend the LEDA arrangement algorithm to conics. Arranging 60 random conics with 50 bit coefficients takes 49 seconds on a 846MHz Pentium 3. Eigenwillig et al [3] extend the LEDA arrangement algorithm to cubics. Arranging 60/90/120/250 random cubics with 100 bit coefficients takes 20/60/110/180 seconds on a 1.2GHz Pentium 3. Geismann et al [5] compute arrangements of special quartics (used to compute arrangements of 3D quadratics) with a sweep algorithm. Arranging 3 quartics with 30 bit coefficients takes 186 seconds on a Pentium 700. Wolpert [19] computes arrangements of nonsingular algebraic curves by an unimplemented sweep algorithm.

Mourrain et al [16] compute arrangements of 3D quadratics by an unimplemented plane sweep algorithm. Geismann et al [5, 17] compute arrangements of 3D quadratics. Keyser et al [8] compute arrangements of low-degree sculpted solids without degeneracies.

3 Input specification

The inputs to the sweep algorithm are curve segments and a crossing module. We develop an algorithm for unbounded curve segments in Section 4 and extend it to bounded monotonic segments in Section 5. This section specifies the algorithm inputs.

3.1 Segments

A curve segment is a connected submanifold of the xy plane that is the graph of a continuous function $y = f(x) : \mathrm{I} \rightarrow \Re$ with $\mathbf{I}$ an interval. Let $\min_x(f) = \min(\mathbf{I}(f))$ and $\max_x(f) = \max(\mathbf{I}(f))$. An unbounded segment has $\min_x(f) = -\infty$ and $\max_x(f) = \infty$, a semi-bounded segment has finite $\max_x(f)$, and a bounded segment has finite $\min_x(f)$ and $\max_x(f)$.

The crossing module takes segments f and g and returns a *crossing list* $\langle f, g, r_1, r_2, \ldots, r_m\rangle$. The r_i are the crossing intersection points where $f(r_i) = g(r_i)$ and $f(x) - g(x)$ changes sign. The function $\mathrm{sgn}(f, g, x)$ expresses the sign of $f(x) - g(x)$ for $x \in \mathrm{I}(f) \cap \mathrm{I}(g)$, equivalently the vertical order of $f(x)$ and $g(x)$, according to the crossing list. Its value is -1 for $x < r_1$, 0 for $x = r_1$, 1 for $r_1 < x < r_2$, and so on. It is evaluated in $\log m$ time by binary search of the crossing list. If $f(x) > g(x)$ for $x < r_1$, the module returns the crossing list $\langle g, f, r_1, r_2, \ldots, r_m\rangle$. In this case, we evaluate $\mathrm{sgn}(f, g, x)$ by the rule $\mathrm{sgn}(f, g, x) \equiv -\mathrm{sgn}(g, f, x)$.

The function $\mathrm{sgn}_+(f, g, r)$ denotes the right limit of $\mathrm{sgn}(f, g, x)$ at $x = r$, and $\mathrm{sgn}_-(f, g, r)$ denotes the left limit. If x is not a crossing, $\mathrm{sgn}(f, g, x) = \mathrm{sgn}_+(f, g, x) = \mathrm{sgn}_-(f, g, x)$. Like sgn, sgn_+ and sgn_- are evaluated in $\log m$ by binary search of the crossing list.

Inconsistency The crossing module is inconsistent for segments f, g, h at x when it outputs crossing lists for which $\mathrm{sgn}_+(f, g, x) = \mathrm{sgn}_+(g, h, x) = \mathrm{sgn}_+(h, f, x)$. The segments are in cyclic vertical order according to the crossing lists, whereas the actual segments are linearly ordered. The inconsistency occurs throughout the open interval (r, s) that is bounded by the closest crossings/endpoints $r \leq x$ and $s > x$ in the three lists. We call such an interval an *inconsistency*. The sweep algorithms in Sections 4 and 5 remove inconsistencies by adding and deleting crossings.

Assumptions

1. Crossing list construction is linear in the maximum number of crossings.

2. The crossing module is *strongly ϵ-accurate* according to the following definition. Our formal results are independent of ϵ: ϵ is not a input parameter to our algorithms, and they are correct for any value of ϵ. They are useful because ϵ is tiny in practice.

Definition 3.1 *Let curves f, g have crossings $r_1, r_2, \ldots, r_m$ and a corresponding sign function* $\mathrm{sgn}(f, g, x) \equiv -\mathrm{sgn}(g, f, x)$.

1. sgn *is* **δ_y-accurate at** x *if* $\mathrm{sgn}(f, g, x) \leq 0$ *implies* $f(x) - g(x) \leq \delta_y$.

2. sgn *is* **strongly (δ_x, δ_y)-accurate** *if it is δ_y-accurate for all $x \in [r_i + \delta_x, r_{i+1} - \delta_x]$ for $i = 0, 1, 2, \ldots, m$, where* $r_0 = \min(\mathbf{I}(f) \cap \mathbf{I}(g)) - \delta_x$ *and* $r_{m+1} = \max(\mathbf{I}(f) \cap \mathbf{I}(g)) + \delta_x$.

3. sgn *is* **strongly δ-accurate** *if it is strongly (δ_x, δ_y)-accurate and $\delta = \delta_x + \delta_y$.*

The notation $[r_i + \delta, r_{i+1} - \delta]$ is the interval $[r_i, r_{i+1}]$ shrunk by $\delta > 0$ from each end, but if $r_i + \delta > r_{i+1} - \delta$, it denotes the single value $(r_i + r_{i+1})/2$. This restriction prevents an accurate crossing module from inserting spurious pairs of nearby roots in its output.

Crossing lists cannot model curves that intersect tangentially or over an interval. Nevertheless, these cases are allowed in the input. The crossing module maintains ϵ-accuracy by omitting tangential intersections and assigning non-zero signs on intervals of intersection.

3.2 Monotonic segments

A segment is monotonic when $f : \mathrm{I}_x(f) \to \mathrm{I}_y(f)$ is bijective, where $\mathrm{I}_x(f)$ (formerly $\mathrm{I}(f)$) and $\mathrm{I}_y(f)$ denote the domain and range. Let $\min_y(f) = \min(\mathrm{I}_y(f))$ and $\max_y(f) = \max(\mathrm{I}_y(f))$. The segment connects two diagonal vertices of its bounding rectangle $\mathrm{rect}(f) = \mathrm{I}_x(f) \times \mathrm{I}_y(f)$. An increasing segment connects $\mathrm{tail}(f) = (\min_x(f), \min_y(f))$ to $\mathrm{head}(f) = (\max_x(f), \max_y(f))$ and a decreasing segment connects $\mathrm{tail}(f) = (\min_x(f), \max_y(f))$ to $\mathrm{head}(f) = (\max_x(f), \min_y(f))$.

The monotonic segment crossing module takes f and g and returns a list of crossing *points* $r_1, r_2, \ldots, r_m \in \mathrm{rect}(f) \cap \mathrm{rect}(g)$ whose x-coordinates and y-coordinates are the crossing values for crossing lists in each coordinate direction. Let $\langle f, g, r_{1x}, r_{2x}, \ldots, r_{mx}\rangle$ be the x crossing list. If f is increasing, the y crossing list is $\langle g, f, r_{1y}, r_{2y}, \ldots, r_{my}\rangle$. If f is decreasing, it is $\langle f, g, r_{my}, r_{m-1y}, \ldots, r_{1y}\rangle$. $\mathrm{sgn}_x(f, g, x)$ and $\mathrm{sgn}_y(f, g, y)$ are the sign functions of the x and y crossing lists.

Suppose p is an endpoint of f. Let g be another segment such that $\mathrm{I}_x(g)$ properly overlaps $\mathrm{I}_x(f)$ and $p_x \in \mathrm{I}_x(g)$. Using sgn_x, we can tell whether p is above or below g. If $p = \mathrm{tail}(f)$, examine $\mathrm{sgn}_{x+}(f, g, p_x)$, and if $p = \mathrm{head}(f)$, examine $\mathrm{sgn}_{x-}(f, g, p_x)$. The value -1 mean p is below, and the value $+1$ means p is above. Similarly, we can use sgn_y to determine if p is left or right of g such that $p_y \in \mathrm{I}_y(g)$. If $p \in \mathrm{rect}(g)$ and g is increasing, p is above g if and only if p is left of g and similarly if g is decreasing and/or p is below.

Assumptions

1. Crossing list construction is linear in the maximum number of crossings.
2. The crossing module is strongly ϵ-accurate in x and y.
3. (General position) No three segments share an endpoint. No endpoint lies on another segment. If two segments share a tail p, no segment endpoint q can satisfy $q_x < p_x$ and $q_y = p_y$.
4. If p is an endpoint of f and g, the f, h and g, h crossing lists agree on the left/right and above/below status of p with respect to h.
5. If $\min_x(f) = \min_x(g)$ and $\mathrm{tail}(f)_y < \mathrm{tail}(g)_y$, then $\mathrm{sgn}_{x+}(f, g, \min_x(f)) = -1$, and similarly for $\max_x$ and for the y-direction.

4 Unbounded segments

This section presents a sweep algorithm for unbounded curve segments, which need not be monotonic. The algorithm uses a vertical sweep line $x = r$ and three data structures.

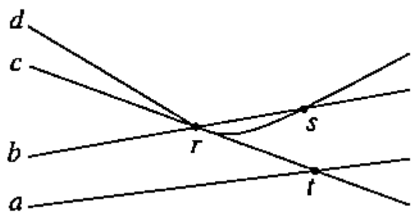

Figure 1: Sample sweep events at $x = r$.

1. a list S of segments called the *sweep list* representing the order of the segments from lowest to highest along the sweep line. S is implemented as a red-black binary tree whose inorder traversal is the list order. The successor and predecessor of f in S are denoted $\text{succ}(f)$ and $\text{pred}(f)$.

2. a priority queue P of events ordered by x: $\text{insert}(f, r)$, insert f into S at $x = r$, or $\text{swap}(f, g, r)$, swap f with g in S at $x = r$. P is implemented as a standard heap data structure with ties (equal r) broken arbitrarily.

3. a set C of output crossing lists. C is implemented as a hash tree with key an unordered pair of segments.

For sweep line $x = r$, S is *locally consistent* if $\text{sgn}_+(f, g, r) = -1$ for each pair $f, g \in S$ such that $g = \text{succ}(f)$. This invariant holds after all events at r are processed.

The algorithm is as follows. Initialize P with $\text{insert}(f_i, -\infty)$ for $i = 1, 2, \ldots, n$. Initialize S and C to be empty. While P is not empty, dequeue an event and process it.

$\text{insert}(f, r)$: Set the sweep line to $x = r$ and insert f into S. If S is empty, set $\text{root}(S) = f$, else if $\text{sgn}_+(f, \text{root}(S), r) = -1$, insert f recursively into $\text{left}(S)$, else insert f recursively into $\text{right}(S)$. Rebalance S as necessary. If f is not the first element of S, then $\text{pred}(f), f$ is a *newly adjacent pair*. If f is not the last element of S, then $f, \text{succ}(f)$ is a newly adjacent pair. For each newly adjacent pair u, v, add $\langle u, v\rangle$ to C and invoke the crossing module for u, v to obtain the *input* crossing list $\langle u, v, r_1, r_2, \ldots, r_m\rangle$. Enqueue into P each corresponding *input* swap event for u, v: $\text{swap}(f, g, r_1)$, $\text{swap}(g, f, r_2)$, $\text{swap}(f, g, r_3)$, etc.

$\text{swap}(f, g, r)$: Set the sweep line to $x = r$ and check if $g = \text{succ}(f)$ in S. If not, discard this event. Otherwise, swap the positions of f and g in S. Look up the crossing list for f, g in C and append r to it. If g is not the first element in S, the pair $\text{pred}(g), g$ is newly adjacent. Similarly, $f, \text{succ}(f)$ if f is not the last element. For each newly adjacent pair u, v, if C does not contain a crossing list for u, v, add $\langle u, v\rangle$ to C, invoke the crossing module on u, v, and enqueue into P input swap events for u, v with $r_i > r$. If $\text{sgn}_+(u, v, r) = 1$ (u, v not locally consistent), enqueue $\text{swap}(u, v, r)$ into P. We call this event an *immediate swap*.

Figure 1 illustrates the algorithm. The S order is a, b, c, d when $\text{swap}(b, c, r)$ is processed. The swap is executed because $c = \text{succ}(b)$, r is appended to the $\langle b, c\rangle$ crossing list in C, b and c are swapped in S, and the S order becomes a, c, b, d. The freshly adjacent pairs are a, c and b, d. An immediate $\text{swap}(b, d, r)$, is enqueued because $\text{sgn}_+(b, d, r) = 1$. These segments were inserted in S in sgn order and crossed at $r' < r$, but the crossing module calculated $r < r'$, so their first adjacency occurred after they should have swapped. Regular $\text{swap}(d, b, s)$ and $\text{swap}(a, c, t)$ events are also enqueued. The immediate swap is executed next, then the swap at s, then the swap at t.

After the sweep is complete, the state of S and C corresponds to a new crossing module. To obtain the crossing list for f, g, look up the pair in C and return its entry if it exists. If f, g does not have an entry in C, return $\langle f, g\rangle$ if f comes before g in S and $\langle g, f\rangle$ if it comes after. Let sgn' be the new *output* sign function for this new crossing module.

4.1 Correctness and Complexity

Lemma 4.1 *Immediately after* $\mathrm{insert}(f, r)$ *is executed,* f *is locally consistent with respect to its neighbors:* $\mathrm{sgn}_+(\mathrm{pred}(f), f) = -1$ *if* f *is not first and* $\mathrm{sgn}_+(f, \mathrm{succ}(f)) = -1$ *if* f *is not last. Immediately after* $\mathrm{swap}(f, g, r)$ *is executed,* $\mathrm{sgn}_+(g, f, r) = -1$.

Proof. The tree insertion preserves local consistency because if f is not the first element in S, then $\mathrm{pred}(f)$ is the root e of the lowest subtree S' containing f in $\mathrm{right}(S')$. Since the algorithm inserted f in $\mathrm{right}(S')$, $\mathrm{sgn}(f, e, r) = 1$ and thus $\mathrm{sgn}(e, f, r) = -1$. Similarly, the successor.

The sign of $\mathrm{sgn}_+(f, g, x)$ alternates at crossings of f, g: $\mathrm{sgn}_+(f, g, r_1) = 1$, $\mathrm{sgn}_+(f, g, r_2) = -1, \ldots$, and hence $\mathrm{sgn}_+(g, f, r_1) = \mathrm{sgn}_+(f, g, r_2) = \mathrm{sgn}(g, f, r_3) = \cdots = -1$. Therefore, the input swaps $\mathrm{swap}(f, g, r_1)$, $\mathrm{swap}(g, f, r_2)$, $\mathrm{swap}(f, g, r_3)$, $\ldots$ satisfy the claim. An immediate swap $\mathrm{swap}(f, g, r)$ is enqueued only if $\mathrm{sgn}_+(f, g, r) = 1$ and therefore $\mathrm{sgn}_+(g, f, r) = -1$. □

Let $S(x)$ denote the the state of S after all events at $r \leq x$ are processed but before any event at $s > x$ is processed.

Lemma 4.2 $S(x)$ *is locally consistent for all* x.

Proof. It suffices to consider $x = r$ at which an event occurs. Consider the different ways a pair f, g can become adjacent in S. By the previous lemma, the events $\mathrm{insert}(f, r)$, $\mathrm{insert}(g, r)$, or $\mathrm{swap}(g, f, r)$ have $\mathrm{sgn}_+(f, g, r) = -1$ as a post-condition. If f, g became newly adjacent as a result of another event at $x = r$, consider the last time they became newly adjacent. If $\mathrm{sgn}(f, g, r) = 1$, an immediate swap $\mathrm{swap}(f, g, r)$ would have been enqueued and eventually processed, making the order g, f instead of f, g. Therefore, $\mathrm{sgn}(f, g, r) = -1$.

Finally, if f, g were adjacent prior to any event at $x = r$ being processed, assume inductively that $S(q)$ is locally consistent, where $q < r$ is the previous event value. The pair f, g cannot have an input crossing at $r_i \in (q, r]$ because the algorithm checks every newly adjacent pair for future swaps and enqueues them. The swap $\mathrm{swap}(f, g, r_i)$ would have been executed and made the order g, f instead of f, g at $x = r$. Therefore, $\mathrm{sgn}_+(f, g, r) = \mathrm{sgn}(f, g, r) = \mathrm{sgn}_+(f, g, q)$. By the inductive assumption, $\mathrm{sgn}_+(f, g, q) = -1$. □

Lemma 4.3 $\mathrm{sgn}'_+(f, g, x) = -1$ *if and only if* f *precedes* g *in* $S(x)$.

Proof. If the pair f, g has no entry in C, the output crossing list is determined by their order in $S(\infty)$, the final S. Since f, g does not appear in C, they did not swap, and so their order in $S(x)$ is the same.

Suppose f, g has the entry $\langle f, g, r_1, r_2, \ldots, r_m\rangle \in C$. (If $\langle g, f, r_1, r_2, \ldots, r_m\rangle \in C$, switch the roles of f and g and use the fact that $\mathrm{sgn}'(f, g, x) \equiv -\mathrm{sgn}'(g, f, x)$.) Suppose $r_1, \ldots, r_i \leq x$ and $x < r_{i+1}, \ldots, r_m$. By definition, the value of $\mathrm{sgn}'(f, g, x)$ is -1 if i is even and 1 if i is odd. However, each r_j corresponds to a swap in S executed at $x = r_j$. Therefore, f comes before g in $S(x)$ if and only if i is even. □

Lemma 4.4 *If* $\mathrm{sgn}_+(f,g,x) \neq \mathrm{sgn}'_+(f,g,x)$, *there exist segments* a,b,c,d *such that* f,a,b *and* g,c,d *are inconsistent at* x.

Proof. Suppose $\mathrm{sgn}_+(f,g,x) = 1$ and $\mathrm{sgn}'_+(f,g,x) = -1$. By the previous lemma, f comes before g in $S(x)$. Let L be the sublist of $S(x)$ from f to g. By Lemma 4.2, if $h \in L$ and $h \neq g$, then $\mathrm{sgn}_+(h, \mathrm{succ}(h), x) = -1$.

Repeat the following step: if the neighbors of some $h \in L$, $h \neq f$ and $h \neq g$, satisfy $\mathrm{sgn}_+(\mathrm{pred}(h), \mathrm{succ}(h), x) = -1$, remove h from L. (Here we are calculating $\mathrm{pred}(h)$ and $\mathrm{succ}(h)$ relative to current state of L.) Do this in any order until there are no such h. Every three consecutive elements in L must now be inconsistent at x. We call such an L a *minimal locally consistent list* from f to g at x. Lemma 4.2 implies that L contains three or more elements because $\mathrm{sgn}_+(f,g,x) = 1$ and f precedes g in L. The first three elements, f,a,b are inconsistent at x, as are the last three elements c,d,g (although these might be the *same* three elements if $|L| = 3$). □

Theorem 4.5 *1) The number of segment endpoints (at* $\pm\infty$*) plus output segment crossings is* $V = 2n + N + \min(3kn, 0.5n^2)$ *and the running time is* $\mathrm{O}(V \log)$, *where* N *is the number of input crossings and* $k = \mathrm{O}(n^3)$ *is the number of input inconsistencies. 2) The output is consistent. 3) If the input has no inconsistencies, then* $\mathrm{sgn}' \equiv \mathrm{sgn}$.

Proof.

1) There are n insert events and N input swap events. The next paragraph shows that at most V events are executed, as opposed to discarded. Since each executed event can enqueue no more than two immediate swap events, the total number of events processed is $n + N + 2V \leq 3V$. Therefore, there are $\mathrm{O}(V)$ tree and queue operations, for a total running time of $\mathrm{O}(V \log n)$.

All n insertions are executed. A pair f, g with input crossing list $\langle f, g, r_1, \ldots, r_m \rangle$ can swap at most once per interval $[-\infty, r_1), [r_1, r_2), \ldots, [r_m, \infty)$ because $\mathrm{sgn}_+(f,g,x)$ is invariant on each interval. If a swap is executed, sgn_+ will have the correct value, so another swap will not be enqueued in the interval. Nominally, f and g should swap m times, but if they have an extra swap in $[-\infty, r_1)$, then g must come before f in $S(-\infty)$. By Lemma 4.3, $\mathrm{sgn}'_+(g, f, -\infty) = -1$, but according to the input crossing list, $\mathrm{sgn}_+(g, f, -\infty) = 1$. By Lemma 4.4, there must be an inconsistency involving f. Each of k inconsistencies involves three segments, and each segment is in $n-1$ pairs. Therefore, there are at most $3k(n-1)$ extra swaps. Other the other hand, there are no more than $n(n-1)/2$ pairs. Therefore the number of extra swaps is at most $\min(3kn, 0.5n^2)$, and the total number of executed events is $n + N + \min(3kn, 0.5n^2)$.

There are $\mathrm{O}(n^3)$ triples of segments. If the maximum number of crossings per pair is m, there are $\mathrm{O}((nm)^3)$ possible inconsistencies, since each inconsistency occurs over a maximal interval that is free of f, g, h crossings. We assume that m is a constant to obtain the simplified result $k = \mathrm{O}(n^3)$.

2) Given x and and a triple of segments f, g, h, suppose $\mathrm{sgn}'(f,g,x) = \mathrm{sgn}'(g,h,x) = -1$. By Lemma 4.3, f precedes g and g precedes h in $S(x)$. Therefore f precedes h, and $\mathrm{sgn}'(f,h,x) = -1$. Therefore, $\mathrm{sgn}'(h,f,x) \neq -1$, and there is no inconsistency.

3) If $\mathrm{sgn} \not\equiv \mathrm{sgn}'$, there exists x and segments f, g such that $\mathrm{sgn}(f,g,x) \neq \mathrm{sgn}'(f,g,x)$. By Lemma 4.4 there must be an inconsistency. The claim is the contrapositive. □

4.2 Accuracy

The input crossing module is assumed strongly ϵ-accurate according to Definition 3.1. The output is trivially strongly ϵ-accurate when $k = 0$ by Theorem 4.5. It might be possible to prove that sgn' is always strongly δ-accurate for small δ, perhaps $O(k\epsilon)$. We prove a weaker claim: if the input is strongly accurate for small ϵ, the output is *weakly* accurate for small δ, as defined below. This claim suffices for our main purpose, which is to prove realizability.

Definition 4.1 *Let segments f, g have crossings $r_1, r_2, \ldots, r_m$ and a corresponding sign function* $\mathrm{sgn}(f, g, x)$.

1. sgn *is* **weakly (δ_x, δ_y)-accurate** *iff for all $x \in \mathrm{I}(f) \cap \mathrm{I}(g)$ there exists x' such that $|x' - x| \le \delta_x$ and* sgn *is δ_y-accurate at x' (Definition 3.1, Part 1).*

2. sgn *is* **weakly δ-accurate** *if it is weakly (δ_x, δ_y)-accurate and $\delta = \delta_x + \delta_y$.*

Strong (δ_x, δ_y)-accuracy implies weak (δ_x, δ_y)-accuracy because each $x \in [r_i, r_{i+1}]$ lies within δ_x of some $x' \in [r_i + \delta_x, r_{i+1} - \delta_x]$. The converse is false, hence the terms strong and weak.

Theorem 4.6 *If* sgn *is strongly ϵ-accurate with $k > 0$ inconsistencies,* sgn' *is weakly $\epsilon + 3k\epsilon$-accurate.*

Proof. Suppose $\mathrm{sgn}'(f, g, x) = -1$ but $\mathrm{sgn}(f, g, x) = 1$. Let $L = \langle h_1 = f, h_2, h_3, \ldots, h_l = g\rangle$ be a minimal locally consistent list from f to g as constructed in the proof of Lemma 4.4. Suppose x is far from any crossing. Since $h_i(x) - h_{i+1}(x) \le \epsilon$ for $1 \le i \le l-1$, it follows that $f(x) - g(x) = h_1(x) - h_l(x) \le (l-1)\epsilon$. Now, every three consecutive segments in L are inconsistent. Therefore $k \ge l - 2$ and $(l-1)\epsilon \le \epsilon + k\epsilon$. Therefore, sgn is weakly $(0, \epsilon + k\epsilon)$-accurate at x.

This proof fails when x is near one of the boundaries of an inconsistency. Let $[r_{\min}, r_{\max}]$ be an interval of inconsistency. We need to find an $x' < x$ that is near x and that avoids the intervals $[r_{\min}, r_{\min} + \epsilon]$ and $[r_{\max} - \epsilon, r_{\max}]$ for each inconsistency. The combined length of the excluded intervals is $2k\epsilon$, so x' can be chosen within $2k\epsilon$ of x.

If $\mathrm{sgn}(f, g, x') = 1$ and $\mathrm{sgn}'(f, g, x') = -1$, the error at x' is $\delta_y = \epsilon + k\epsilon$ by the argument in the first paragraph, since x' is far from any relevant crossing. From the previous paragraph, $\delta_x = 2k\epsilon$, and sgn' is weakly $(2k\epsilon, \epsilon + k\epsilon)$-accurate.

If $\mathrm{sgn}(f, g, x') = -1$, there exists x'' within ϵ of x' such that $f(x) - g(x) \le \epsilon \le k\epsilon$, since $k \ge 1$. It follows that $|x'' - x| \le |x'' - x'| + |x' - x| \le \epsilon + 2k\epsilon$. So sgn' is weakly $(\epsilon + 2k\epsilon, k\epsilon)$-accurate.

If $\mathrm{sgn}'(f, g, x') = 1$, there must be an event $\mathrm{swap}(g, f, r)$ for $x' < r < x$, which implies that $\mathrm{sgn}_+(f, g, r) = -1$. Move x' just to the right of r and apply the result of the previous paragraph. If x' were *greater* than x, this reasoning would not work: $\mathrm{swap}(f, g, r)$ does *not* imply that $\mathrm{sgn}_-(f, g, r) = -1$ because the algorithm cannot advance crossings, just postpone them. □

4.3 Realizability

Let $\mathrm{dist}(p, f)$ denote the distance from point p to segment f: the minimum over x of $|p - (x, f(x))|$.

Definition 4.2 *Given a segment f, real x, and $\delta \geq 0$, define,*

$$\begin{aligned} \text{near}(f, x, \delta) &= \{y \mid \text{dist}((x, y), f) \leq \delta\} \\ \text{below}(f, x, \delta) &= \text{near}(f, x, \delta) \cup [-\infty, f(x)], \\ \text{above}(f, x, \delta) &= \text{near}(f, x, \delta) \cup [f(x), \infty]. \end{aligned}$$

In effect, $\text{near}(f, x, \delta)$ denotes the interval of y values near $f(x)$, taking into account the slope of f: a steep segment will have a longer y interval near it.

Definition 4.3 *Given a sign function* $\text{sgn}(f, g, x)$ *on segments f and g:*

1. sgn *is* δ-**realizable at** x *if* $\text{sgn}(f, g, x) \leq 0$ *implies* $\text{above}(f, x, \delta) \cap \text{below}(g, x, \delta) \neq \emptyset$;
2. sgn *is* δ-**realizable** *if* $\text{sgn}(f, g, x)$ *is δ-realizable for all x.*

Lemma 4.7 *Weak (δ_x, δ_y)-accuracy implies $\delta_x + \delta_y$-realizability.*

Proof. Suppose $\text{sgn}(f, g, x) \leq 0$, and let x' satisfy the definition of weak (δ_x, δ_y)-accuracy. The set of points $\{x'\} \times \text{near}(f, x', \delta_y)$ lies within δ_y of f. The set of points $\{x\} \times \text{near}(f, x', \delta_y)$ is $|x - x'| \leq \delta_x$ to the left or right and therefore lies with $\delta_x + \delta_y$ of f. Hence,

$$\text{near}(f, x', \delta_y) \subseteq \text{near}(f, x, \delta_x + \delta_y).$$

It follows that,

$$\text{above}(f, x', \delta_y) \subseteq \text{above}(f, x, \delta_x + \delta_y),$$

and similarly for below and for g. We have $f(x') \in \text{near}(f, x', \delta_y) \subset \text{above}(f, x', \delta_y)$ by the definition of near and $f(x') \in \text{below}(g, x', \delta_y)$ by the definition of (δ_x, δ_y)-accurate. Therefore, $\text{above}(f, x', \delta_y) \cap \text{below}(g, x', \delta_y) \neq \emptyset$. By the subset relationships shown above, $\text{above}(f, x, \delta_x + \delta_y)) \cap \text{below}(g, x, \delta_x + \delta_y)) \neq \emptyset$. □

Theorem 4.8 *Suppose* sgn *is consistent and δ-realizable at x for segments $g_1, g_2, \ldots, g_n$ such that* $\text{sgn}(g_i, g_j, x) = -1$ *for $1 \leq i < j \leq n$. There exist $y_1 \leq y_2 \leq \cdots \leq y_n$ such that for $1 \leq i \leq j \leq n$, $y_i \in \text{below}(g_j, x, \delta)$ and $y_j \in \text{above}(g_i, x, \delta)$.*

Proof. For $1 \leq i \leq n$, define

$$Y_i = \bigcap_{h=1}^{h=i} \text{above}(g_h, x, \delta) \cap \bigcap_{j=i}^{j=n} \text{below}(g_j, x, \delta).$$

The set Y_i is an intersection of intervals, and each pair of intervals has non-empty intersection for the following reasons. A pair of above intervals have non-empty intersection, and similarly each pair of below intervals. By definition of near, $\text{above}(g_i, x, \delta) \cap \text{below}(g_i, x, \delta) = \text{near}(g_i, x, \delta)$. By definition of δ-realizable, for $h \neq j$, $\text{above}(g_h, x, \delta) \cap \text{below}(g_j, x, \delta) \neq \emptyset$.

Since each pair of intervals intersects, all pairs must have a common element (interval version of Helley's theorem), and Y_i is non-empty. Let $y_i = \min(Y_i)$. These y_i satisfy the final clause of the theorem. All we need to show is that $y_i \leq y_{i+1}$. However, the construction of Y_{i+1} has $\text{below}(g_{i+1}, x, \delta)$ replaced by $\text{above}(g_{i+1}, x, \delta)$, which cannot decrease the minimum. □

Definition 4.4 *An* **x-monotonic** *curve is a continuous curve whose intersection with each vertical line $x = a$ is either a single point or, for a finite number of vertical lines, a vertical line segment, pq with $p_x = q_x$.*

Just as a curve segment is equivalent to a continuous function $y = f(x)$, an x-monotonic curve is equivalent to a piecewise continuous function $y = \hat{f}(x)$ whose discontinuities are filled in with vertical lines segments.

Definition 4.5 *A set $\hat{f}_1, \hat{f}_2, \ldots, \hat{f}_n$ of x-monotonic curves* **realizes** *a sign function* sgn *if for $1 \leq i < j \leq n$ and for every x for which $\hat{f}_i(x)$ and $\hat{f}_j(x)$ are continuous the following holds:* $\mathrm{sgn}(f_i, f_j, x) \leq 0$ *implies $\hat{f}_i(x) \leq \hat{f}_j(x)$ and* $\mathrm{sgn}(f_i, f_j, x) \geq 0$ *implies $\hat{f}_i(x) \geq \hat{f}_j(x)$.*

The x-monotonic curves cross where sgn says they do, but they can overlap along a vertical line segment at the crossing.

Theorem 4.9 *Let* sgn *on $f_1, f_2, \ldots, f_n$ be consistent and δ-realizable. There exist x-monotonic curves $\hat{f}_1, \hat{f}_2, \ldots, \hat{f}_n$ that realize* sgn *such that $\hat{f}_i$ lies within δ of f_i for $1 \leq i \leq n$ (for all x,* $\mathrm{dist}((x, \hat{f}_i(x)), f_i) \leq \delta$*).*

Proof. Given a curve g, we state as mathematical fact that the set of points

$$\{p \,|\, p \in \mathrm{below}(g) \text{ and } \mathrm{dist}(p, g) = \delta\}$$

is a continuous curve: the *lower δ-level set* of g.

For each x not a crossing, $\hat{f}(x)$ takes on the value of y_i given in Theorem 4.8, where $f = g_i$ (f is the ith curve in the vertical order at x). We claim that the resulting function is continuous away from crossings. The construction of the lemma sets $\hat{f}(x)$ equal to upper envelope (maximum) of lower δ-level sets of $g_1, g_2, \ldots, g_i$. As long as $f = g_i$ does not cross another curve, the list $g_1, g_2, \ldots, g_{i-1}$ might change order, but the set $g_1, g_2, \ldots, g_i$ will stay the same. The maximum of a set of continuous functions is continuous.

At a crossing $x = r$ (indeed at any x), the limits $\hat{f}_-(r)$ and $\hat{f}_+(r)$ of $\hat{f}(x)$ from the left and right both lie in $\mathrm{above}(f, r, \delta)$ and $\mathrm{below}(f, r, \delta)$. Therefore they and the interval between them lies in $\mathrm{near}(f, r, \delta)$, and thus all of $\hat{f}$ (even its vertical segments) lie within δ of f. □

Corollary 4.10 *The output of the sweep algorithm is a crossing module for a consistent sign function realized by x-monotonic curves $\hat{f}_1, \hat{f}_2, \ldots, \hat{f}_n$ that lie with $\epsilon + 3k\epsilon$ of $f_1, f_2, \ldots, f_n$.*

Proof. Part 2 of Theorem 4.5 implies that the output sign function is consistent. Apply Theorem 4.9 to the results of Theorem 4.6 and Lemma 4.7. □

The realization curves have vertical segments, whereas the input curves are monotonic in x. There is no geometric significance to vertical segments. Moreover, the sweep algorithm applies to x-monotonic curves with minor changes. Although the realization curves in the proof overlap over short intervals, this overlap is eliminated by a simple perturbation.

5 Bounded segments

This section describes a sweep algorithm for (bounded or unbounded) monotonic segments that matches the performance of the unbounded segment algorithm. Section 5.1 extends the sweep to semi-bounded segments. Section 5.2 extends it to bounded segments, but without an error bound. Section 5.3 derives an error bound for specialized bounded segments that have a tree structure. Section 5.4 handles monotonic segments by combining these results.

5.1 Semi-bounded segments

The unbounded segment sweep extends to semi-bounded segments with the following changes.

Initialization Add events remove$(f_i, \max_x(f_i))$: remove f_i from S at $x = r$ for $1 \leq i \leq n$ to the initial state of P. Ties are broken arbitrarily, except that removes come before swaps.

Execution Upon dequeuing remove(f, r) from P, delete f from S. If $e = \text{pred}(f)$ and $\text{succ}(f) = g$ in S prior to this deletion, e and g become freshly adjacent. If either $\max_x(e) = r$ or $\max_x(g) = r$ where $r = \max_x(f)$, nothing needs to be done because an upcoming remove event will remove one of these segments. Otherwise, treat the freshly adjacent pair the same as those arising from swaps.

Correctness and Complexity During the sweep, the n additional remove events create up to n additional newly adjacent pairs, which are treated as in the original algorithm. Therefore, there are no additional crossings and the running time is still $O(V \log n)$.

Accuracy The proof of accuracy in Theorem 4.6 employs shifts to the *left,* which presents no problems for segments whose domains are only bounded on the *right*.

5.2 Bounded segments

The algorithm for semi-bounded segments extends to bounded segments.

Initialization No change.

Execution Ties are broken by processing inserts after swaps and removes. The sweep list S is represented as a persistent binary tree. When f, g lack an entry in the output C, their crossing list is constructed based on their initial order at $x = \max(\min_x(f), \min_x(g))$, which is obtained from S in $\log n$ time.

Correctness and Complexity The proofs of correctness and complexity do not depend on having all insertions at $x = -\infty$, and therefore they hold as before.

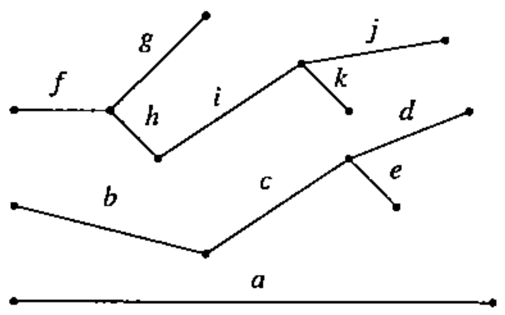

Figure 2: Branching segments.

Accuracy The proof of accuracy fails when the leftward shift reaches the left endpoint of a bounded segment. As far as we know, the numerical error of f with respect to g at $x = \min_x(f)$ may be unbounded, at least for a short x interval.

5.3 Branching segments

Since we could not prove an inconsistency-sensitive error bound for bounded segments, let us generalize semi-bounded segments differently. A forest of binary trees of curve segments defines a set of semi-bounded *branching segments*. The root of each tree is a semi-bounded segment and all other nodes are segments bounded on the left. Since the root of each tree starts to the left at $x = -\infty$ and the children extend to the right to increasing x, the children of a node f are denoted lower(f) and upper(f). However, if there is only one child, it it denoted child(f). If g is a child of f, the pair f, g must satisfy three *continuity conditions*. Let $r = \max_x(f)$.

1. $\min_x(g) = r$.
2. If $\min_x(h) < r < \max_x(h)$, then $\mathrm{sgn}_-(f, h, r) = \mathrm{sgn}_+(g, h, r)$.
3. If g' is a child of f' and $\max_x(f') = r$ ($= \min_x(g')$), then $\mathrm{sgn}_-(f, f', r) = \mathrm{sgn}_+(g, g', r)$.

Each input g is a subset of a semi-bounded segment $\overline{g}$, which denotes the union of g with its ancestors in its tree. If $\overline{g} \neq \overline{h}$, for $x \leq \min(\max_x(g), \max_x(h))$, $\mathrm{sgn}(\overline{g}, \overline{h}, x)$ is well-defined and continuous between segments of $\overline{g}$ and $\overline{h}$ thanks to the three continuity conditions. However, if f is the nearest common ancestor of g and h, then $\overline{g} = \overline{h} = \overline{f}$ for $x \leq \max_x(f)$. Suppose g arises from the lower child of f (lower(f) $\subset \overline{g}$) and h arises from the upper child (upper(f) $\subset \overline{h}$). Then for $x \leq \max_x(f)$, define $\mathrm{sgn}(\overline{g}, \overline{h}, x) = -1$. In other words, we imagine $\overline{g}$ and $\overline{h}$ to contain separate copies of $\overline{f}$ with $\overline{g}$'s copy slightly lower than $\overline{h}$'s copy. The resulting sgn function is strongly ϵ-accurate: for $x \leq \max_x(f)$, $\overline{g}(x) = \overline{f}(x) \leq \overline{f}(x) + \epsilon = \overline{h}(x) + \epsilon$ for any $\epsilon \geq 0$. Figure 2 shows a forest of 11 segments in 3 trees that defines the 6 branching segments a, bcd, bce, fg, $fhij$, and $fhik$.

Initialization Initialize P with insert($f, -\infty$) for every semi-bounded segment and one of the following for each semi-bounded or bounded segment f: remove(f, r), replace(f, child(f), r), or branch(f, lower(f), upper(f), r). These events correspond to f having zero, one, or two children.

Execution Upon dequeuing replace(f, g, r), replace f by g in S. For branch(f, g, h, r), replace f by the sublist g, h in S. Thanks to the three continuity conditions, no immediate swap can occur. Check the (up to three) newly adjacent pairs of segments for future crossings and enqueue swaps as appropriate.

Correctness and Complexity Thanks to the continuity conditions, replace and branch events preserve local consistency. The effect of these events is the same as a remove followed by one or two locally consistent insertions. Therefore, the output complexity and running time remain the same.

Accuracy Since each bounded segment f is now part of a semi-bounded segment $\overline{f}$, the proof of accuracy does not fail when it attempts to shift past the left end of f. Instead, it shifts onto the parent of f in the tree. Hence, the output is $\epsilon + 3k\epsilon$-accurate as in the semi-bounded case.

5.4 Monotonic segments

We combine the prior algorithms to handle monotonic segments. We sweep the segments in the y direction (horizontal sweep line) with the bounded segment algorithm. This sweep enables us to convert the input into branching segments by splitting segments and adding horizontal line segments. We sweep the branching segments to obtain accurate output crossing lists. The inaccuracy of the y sweep manifests itself in erroneous splits, which increase the running time by a constant factor, but is isolated from the output accuracy.

Using sgn_y and the sweep algorithm for bounded segments, conduct a y-sweep of $f_1, f_2, \ldots, f_n$. Do the following for each segment $f \in \{f_1, f_2, \ldots, f_n\}$ whose tail $t = \text{tail}(f)$ is not equal to the head head(g) of some other segment g. When the algorithm encounters $t = \text{tail}(f)$ at the event insert(f) (increasing) or remove(f) (decreasing), record the predecessor g of f in S: g is immediately to the left of f at t but not connected to it. After this sweep, we know (approximately) for each each such segment f the first segment g that a horizontal ray extending leftward from $t = \text{tail}(f)$ will hit. By assumption 3 in Section 3.2, this ray does not pass through a segment endpoint before hitting g.

Each such ray is a semi-bounded constant segment h with $\max_x(h) = t_y$ and t right of g. The latter is true because the sweep is locally consistent and hence $\text{sgn}_y(g, f, t_y) = -1$. Calculate the crossing $x = r$ of intersection between g and h and define $p = (r, t_y)$. Split g into $g_<$ and $g_>$ at p such that $\text{tail}(g_<) = \text{tail}(g)$, $\text{head}(g_<) = \text{tail}(g_>) = p$ and $\text{head}(g_>) = \text{head}(g)$. Set $\min_x(h) = p_x$ (h now connects p to t), but if there was no segment to the left of $t = \text{tail}(f)$ (f was the first element in S at $y = t_y$), h remains semi-bounded. Although one monotonic g might be split by multiple fs, there are at most n splits. The result is up to $2n$ monotonic segments and up to n bounded or semi-bounded constant segments.

Now construct a set of branching segments, based on common endpoints, out of the $3n$ segments. If head(f) is not the tail of another segment, f is a leaf. If $\text{head}(f) = \text{tail}(g)$ for exactly one segment g, then $g = \text{child}(f)$. If $\text{head}(h) = \text{tail}(f) = \text{tail}(g)$ where f and g are not horizontal and $\text{sgn}_{x-}(f, g, \min_x(f)) = -1$, then $f = \text{lower}(h)$ and $g = \text{upper}(h)$. If g was split into $g_<$ and $g_>$ by h at $p = \text{head}(g_<) = \text{tail}(g_>) = \text{tail}(h)$ and if g is decreasing, then $g_> = \text{lower}(g_<)$ and $h = \text{upper}(g_<)$, but if g is increasing, then $h = \text{lower}(g_<)$ and $g_> = \text{upper}(g_<)$. The roots of the

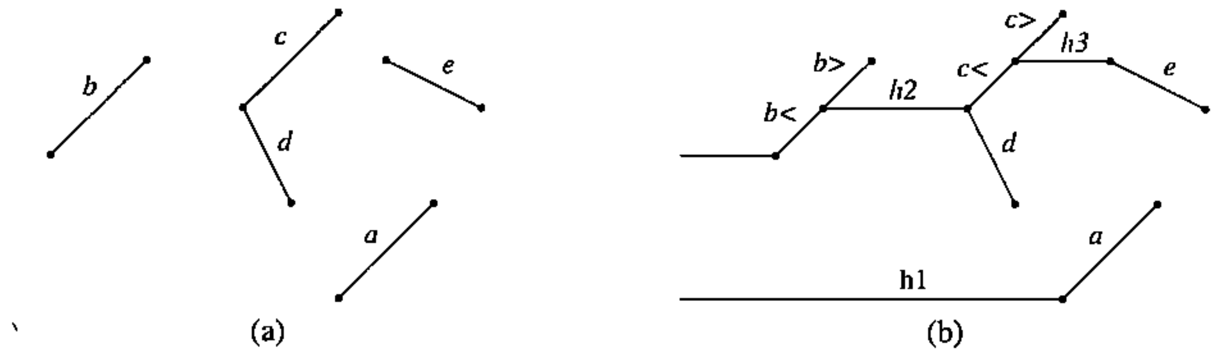

Figure 3: Branching segment construction: (a) input; (b) output.

trees are the set of semi-bounded constant segments: tail rays that do not hit a monotonic segment. Assumptions 3–5 in Section 3.2 imply that the branching segments satisfy the three continuity conditions. Figure 3 shows segments a, b, c, d, e with horizontals h_1, h_2, h_3 that generate branching segments $h_1 a$, $b_< b_>$, $b_< h_2 c_< c_>$, and $b_< h_2 c_< h_3 e$.

Run the sweep algorithm in x for branching segments on the $3n$ segments. In generating the output set of crossing lists, ignore swaps between monotonic and constant segments. Match the remaining swaps to the original monotonic segments (in case one or both monotonic segments were split). The output crossings of the sweep algorithm are consistent and $\epsilon + 3k\epsilon$-realizable. Discarding irrelevant crossings involving added constant segments h does not change this fact.

Analysis The y-sweep has running time in $O(V \log n)$. Its output complexity is $O(V)$, but that does not matter since we only use the y-sweep to calculate n horizontal line segments. In the subsequent x-sweep, a bounded constant segment h (representing horizontal segment pt) might intersect some segment e, but this can only happen if the y-sweep is inconsistent. If e is supposed to be between g and f at $y = t_y$ ($\mathrm{sgn}_y(g, e, t_y) = \mathrm{sgn}_y(e, f, t_y) = -1$) but the y-sweep does not place e there, the y-sweep is inconsistent, and either $\mathrm{sgn}'_y(f, e, t_y) = 1$ or $\mathrm{sgn}'_y(e, g, t_y) = 1$. However, this means we can construct a minimal locally consistent list, and the first (or last) three segments in this chain are inconsistent and involve e. We charge this inconsistency with the extra intersection. Since only three segments belong to an inconsistency, a particular inconsistency can only be charged three times for every h. So the number of spurious intersections of a particular constant segment h with all monotonic segments in the x-sweep is bounded by $3k$, and since there are n added constant segments (horizontal rays or segments), the total number of spurious intersections is at most $3kn$. Thought experiments convince us that a single inconsistency might cause $\Omega(n)$ intersections with horizontal line segments, and so this is probably the best we can prove. On the other hand, a particular constant segment h can intersect at most n monotonic segments and only those whose tails are to the left of $\max_x(h)$ for a total of at most $n(n-1)/2$ crossings. Therefore there are at most $\min(3kn, 0.5n^2)$ extra crossings, and the extra running time is $O(\log n)$ per extra crossing.

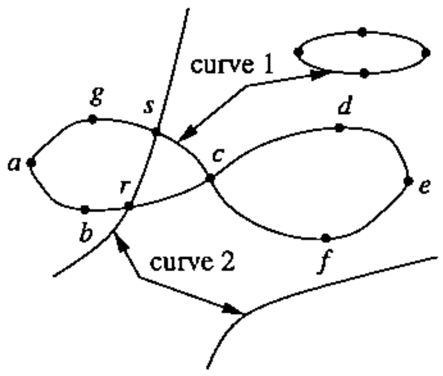

Figure 4: Algebraic curves.

6 Algebraic curves

This section shows that the sweep algorithm applies to algebraic curves. We describe the curves, explain why they satisfy the assumptions of Section 3.2, and describe numerical algorithms for partitioning them into monotonic segments and for computing crossing lists.

Structure A point on the algebraic curve $F(x, y) = 0$ is regular when ∇F is nonzero and is singular otherwise. The regular points partition into 1D manifolds, called branches, that are topological circles or lines. Two branches meet at a singular point. We rule out a *degenerate* singular point whose Hessian matrix has a zero eigenvalue. The curve partitions into monotonic segments that meet at x turning points where $F_y = 0$ or at y turning points where $F_x = 0$.

Figure 4 shows two algebraic curves. Curve 1 consists of three branches: a topological circle and two topological lines (the left/right loops of a horizontal figure 8) that meet at singular point c. The monotonic segments of the figure 8 are ab, bd, de, ag, gf, fe. The segments bd and gf intersect at c. Curve 2 consists of two unbounded monotic segments. The curves intersect at r and s.

Segmentation We employ a sweep to partition a curve into monotonic segments and to compute their vertical order. The events are the turning and singular points, which are computed by solving $F = F_x = 0$ and $F = F_y = 0$. The x coordinates of these points partition the x axis into intervals on which there are a fixed number of monotonic segments. Two segments start or end at an x turning point. One ends and another starts at a y turning point. Two swap vertical order at a singular point. A segment could also start or end at a vertical asymptote, but we ignore this case because it can be eliminated by choosing a random coordinate system.

Sweep updates use equation solving to find the relevant segments. The y values of the sweep segments at $x = x_0$ are computed by solving $F(x_0, y)$, sorting the roots, and assigning the sorted roots to the segments, which are stored in y order. At a start point p (and at $x = -\infty$), we insert the new segment between the segments whose y values bracket p_y. At an end point, we remove the segment whose y value is closest to p_y. At a singular point, we swap the two segments whose y values are closest to p_y.

Solver We solve algebraic equations (one univariate or two bivariate) by the homotopy method [15], which finds all the roots with high probability given a randomized starting point. The rare failures are corrected with restarts. The algorithm converges quadratically at simple roots and the root accuracy is bounded by the condition times the unit roundoff error of about 10^{-16}. At multiple roots, convergence is linear and the accuracy can be lower. For generic input, multiple roots arise solely when we solve $F(x_0, y)$ at a sweep event (x_0, y_0) and y_0 is a double root. We factor $y - y_0$ out of $F(x_0, y)$ and solve the reduced polynomial for the other, simple roots.

Crossing module We compute crossing lists for all segment pairs belonging to the curves F, G when the first list is requested. We solve $F, G = 0$, assign each root to a pair of segments, and construct the lists. The list for segments f, g is defined over the intersection of their x ranges, $\mathbf{I}(f) \cap \mathbf{I}(g)$. The roots assigned to f, g split $\mathbf{I}(f) \cap \mathbf{I}(g)$ into intervals. We compute the vertical order of f and g on each interval by comparing their y values at the midpoint. The roots where the vertical order does not change are dropped and the rest are formed into a crossing list.

A special case occurs when F equals G. In this case, segments f and g cross at singular points that they share in common.

Justifications for Assumptions The following are justifications for the assumptions made in Section 3.2 on page 5.

1. Crossing list construction is linear in the maximum number of crossings, which equals the square of the algebraic degree of the curves by Bezout's theorem. Theory ensures a polynomial bound. The experiments in the next section show that the actual running time increases slightly with algebraic degree. We can enforce the assumption by bounding the degree.

2. The crossings module is strongly ϵ-accurate. The segments can be expressed as analytic functions $f(x)$ and $g(x)$ by the implicit function theorem. Suppose the roots r_i of $h(x) = f(x) - g(x)$ are approximated as s_i. The sgn value is correct outside the intervals $[r_i, s_i]$ (or $[s_i, r_i]$). Consider one interval $[r, s]$. The homotopy method ensures that $|h(s)| < \epsilon$ where ϵ is the floating point rounding unit (about 10^{-16} for ANSII double float). Since $h(r) = 0$, Taylor's theorem yields
$$h(s) = h'(r)e + \frac{h''(r)}{2}e^2 + O(e^3)$$
with $e = s - r$. The quantity $|h'(r)|$ is the tangent of the angle at which h intersects the x axis. Figure 5a shows the generic case where the angle is bounded away from zero, the linear Taylor term dominates, $e \approx h(s)/|h'(r)|$, and $|e| = O(\epsilon)$. The crossing module is ϵ-accurate with $\delta_x = \epsilon$ and $\delta_y = 0$ because every point in $[r, s]$ is within ϵ of the complement of $[r, s]$ where sgn is correct. The error at s, $v \approx eh'(r)$, is unbounded because h can intersect the x axis arbitrarily steeply. Figure 5b shows a non-generic, tangential intersection where $|h'(r)|$ is small, the quadratic term dominates, $e \approx \sqrt{2h(s)/h''(r)}$, and $|e| = O(\sqrt{\epsilon})$. The crossing module is ϵ-accurate with $\delta_x = 0$ and $\delta_y = \epsilon$ because $h(x) = O(\epsilon)$ in $[r, s]$. The error at s is bounded, $v \approx \epsilon$.

3. General position is a standard computational geometry assumption. It holds with probability one for random algebraic curves. Furthermore, we only assume pairwise general position: three or more segments are allowed to intersect at a common point.

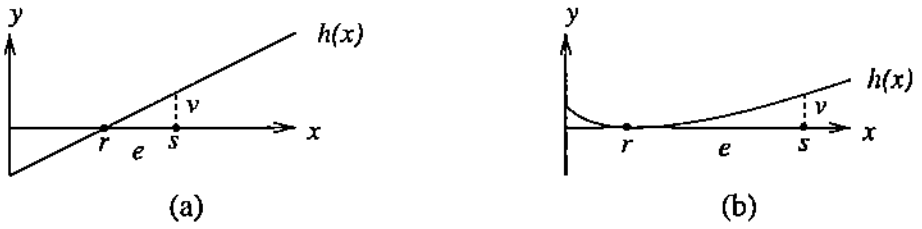

Figure 5: Strong ϵ-accuracy: (a) transverse intersection; (b) tangential intersection.

4. Crossing lists agree on left/right and above/below. We decide these relations once for every point/segment pair and force the crossing lists to be consistent with the decisions.

5. If two segments have equal tail x values, their initial vertical order is determined by their tail y values. We force the crossing lists to respect this condition.

7 Experiments

We estimated the ϵ-accuracy of the crossing module in four ways and obtained identical results: the max ϵ was 10^{-15} for polynomials of degree 1–10 and the average/max were $10^{-12}/10^{-10}$ for degree 11–40. First, we selected random doubles $r_1, \ldots, r_d \in [-1, 1]$, expanded the polynomial $p(x) = (x - r_1) \cdots (x - r_d)$, computed its roots $s_1, \ldots s_n$ with the crossing module, and estimated ϵ at each root as $\min(|r_i - s_i|, |p(s_i)|)$. We generated 100 polynomials for $d = 1, \ldots, 40$. Second, we forced the polynomials to have a double root by setting r_d to r_1. Third, we used polynomials with random coefficients in $[-1, 1]$. This is a general sampling technique, since any polynomial can be expressed in this format by rescaling. Fourth, we used the system $f(x) = g(y) = 0$ with f and g as in the first experiment and with $d = 1, \ldots, 15$. Although the algebraic degree of the roots is d^2, the ϵ values are comparable to those of one degree d polynomial.

We estimated the number of inconsistencies in several types of arrangement problems. We generated 1000 random curves of degree d, constructed their arrangement, and repeated the experiment 50 times. The algorithm never found an inconsistency for $d = 1, 2, 3, 4, 5, 6, 10, 15$ despite the large number of crossings, for example 47511 on a typical set of 1000 quartics. Nor were there any inconsistencies for 1000 random horizontal and vertical line segments, which are common in VLSI, mechanical design, and other applications.

We generated 200 random line segments and perturbed the segment endpoints by a random number in $[-\delta, \delta]$ to obtain 200 pairs of nearly identical segments. We checked all 10.5 million triples of segments for inconsistencies and obtained 0.06% inconsistent for $\delta = 0.001$ and none for $\delta = 0.1, 0.0001, 10^{-6}, 10^{-10}$. We generated 200 near-identical line segments by perturbing the endpoints of a single segment, generated all triples, and obtained no inconsistencies.

We generated 200 random polynomials $y = f(x)$ of degree d then modified their constant terms to make them go through the point $p = (1, 2)$. The polynomial $f(x)$ was replaced with $g(x) = f(x) - f(1)$, so that $g(1) = 0$ except for rounding error. We counted the inconsistencies among all 1.3 million triples of polynomials and obtained none for $d = 1$, 5% for $d = 2$, 39% for $d = 3$, 42% for $d = 4$, 48% for $d = 5$, 54% for $d = 6$, 58% for $d = 10$, and 60% for $d = 15$. The roots were computed by Laguerre's method. We repeated the experiments with random implicit polynomials and obtained 60% inconsistent triples for degree 2 and higher. The roots were computed by the homotopy method. The maximum width of an inconsistent interval was 10^{-10} over all the experiments. The running time per root was roughly constant.

We perturbed the constant terms of the polynomials by a random number in $[-\delta, \delta]$ and obtained 0.003% inconsistencies instead of 48% for $d = 5$ and $\delta = 10^{-8}$, 0.1% instead of 66% for $d = 15$ and $\delta = 10^{-8}$, and 3.5% for $d = 15$ and $\delta = 10^{-10}$.

We conclude that inconsistencies are vanishingly rare in generic input and in many structured, hence degenerate, inputs. The only case where we found many inconsistencies is among triples of curves that almost meet at a point. The curves form a tiny triangle with 4 inconsistent vertex orders and 2 consistent orders. As d increases, the floating point resolution of the triangle decreases until the vertex order becomes essentially random at $d = 15$, that is 60% inconsistent versus 66% for a random choice of 4 out of 6 orders.

Although degenerate, small triangles occur in some applications. For example, consider the layout problem of cutting a maximum number of clothing parts from a strip of fabric. Every part will touch two other parts (or the strip boundary) in an optimal configuration, which implies that three contact curves intersect in every three-part configuration space. In mechanical design, redundancy and symmetry can generate intersecting triples of contact curves. Even so, these degeneracies and the inconsistencies they cause will be confined to small regions, and it is hard to conceive of a practical input for which inconsistencies will even double the running time of the entire arrangement construction.

8 Conclusions

We have presented a robust arrangement algorithm for algebraic curves based on floating point arithmetic. Its performance is analyzed in terms of the number k of combinatorial inconsistencies that occur due to numerical computation. The running time and output size match those of the standard sweep algorithm with exact, unit-cost algebraic computation, plus a term that is linear in k and quadratic in the input size. We have presented extensive experimental evidence that k is zero in generic input and is tiny even in degenerate input, hence that the actual running time matches the standard sweep with floating point algebraic computation. Furthermore, the running time and error bounds appear to depend on *clusters* of inconsistencies. If the k inconsistencies are pairwise separated by more than ϵ in x and y, then their running time cost is linear in k and their accuracy cost is constant.

Inconsistency sensitive analysis is a new computational geometry paradigm that we plan to explore further. The first step is to generalize our algorithm to algebraic curve segments and to construct generic embeddings. Our next goal is to construct and manipulate the configuration spaces of rigid planar parts, which are key to algorithmic part layout, mechanical design, and path planning. Another goal is solid modeling with explicit and implicit surfaces. In both cases, the computational geometry task is to arrange surface patches of high degree.

We also plan to develop iterative algorithms that cascade geometric computations, meaning that the output of each iteration is the input to the next iteration. Many non-geometric numerical algorithms use cascading, for example Newton's method. We believe that geometric algorithms would also use cascading extensively if there were an effective way to implement it. For example, Milenkovic uses cascaded numerical geometric operations in part layout [14, 11, 13]. However, one can construct any algebraic expression by cascading two simple geometric constructions: (1)

join two points to form a line and (2) intersect two lines [1, 12]. This suggests that exact geometric cascading is as hard as exact scientific computing, which is untenable. Inconsistency sensitive algorithms could make cascading practical by replacing this exponential factor with a small constant.

Acknowledgments

Research supported by NSF grants IIS-0082339 and CCR-0306214, and by the Purdue University Center for Graphics, Geometry, and Visualization.

References

[1] Behnke, Bachmann, Fladt, and Kunle. *Fundamentals of Mathematics, Volume II: Geometry.* MIT Press, Cambridge, MA, 1974.

[2] Eric Berberich, Arno Eigenwillig, Michael Hemmer, Susan Hert, Kurt Mehlhorn, and Elmar Schömer. A computational basis for conic arcs and boolean operations on conic polygons. In Rolf H. Möhring and Rajeev Raman, editors, *Algorithms - Esa 2002, 10th Annual European Symposium*, volume 2461 of *Lecture Notes in Computer Science*, pages 174–186, Rome, Italy, 2002. Springer.

[3] Arno Eigenwillig, Lutz Kettner, Elmar Schömer, and Nicola Wolpert. Complete, exact, and efficient computations with cubic curves. In *Proceedings of the 20th ACM Symposium on Computational Geometry*, 2004.

[4] Eyal Flato, Dan Halperin, Iddo Hanniel, Oren Nechushtan, and Eti Ezra. The design and implementation of planar maps in CGAL. *The ACM Journal of Experimental Algorithmics*, 5(13), 2000.

[5] Nicola Geismann, Michael Hemmer, and Elmar Schömer. Computing a 3-dimensional cell in an arrangement of quadrics: exactly and actually! In *Proceedings of the 17th ACM Symposium on Computational Geometry*, pages 264–273, 2001.

[6] Dan Halperin. Robust geometric computation. In J. E. Goodman and J. O'Rourke, editors, *Handbook of discrete and computational geometry*. CRC Press, Boca Raton, FL, second edition, 2004.

[7] Dan Halperin and Eran Leiserowitz. Controlled perturbation for arrangements of circles. In *Proceedings of the 19th ACM Symposium on Computational Geometry*, pages 264–273, San Diego, 2002.

[8] John Keyser, Tim Culver, Mark Foskey, Shankar Krishnan, and Dinesh Manocha. ESOLID—a system for exact boundary evaluation. *Computer-Aided Design*, 36(2):175–193, 2004.

[9] John Keyser, Tim Culver, Dinesh Manocha, and Shankar Krishnan. Efficient and exact manipulation of algebraic points and curves. *Computer Aided Design*, 32:649–662, 2000.

[10] K. Melhorn and S. Näher. *The LEDA platform for combinatorial and geometric computing.* Cambridge University Press, 1999.

[11] V. J. Milenkovic. Rotational polygon containment and minimum enclosure using only robust 2d constructions. *Computational Geometry: Theory and Applications*, 13:3–19, 1999.

[12] Victor J. Milenkovic. Shortest path geometric rounding. *Algorithmica*, 27(1):57–86, 2000.

[13] Victor J. Milenkovic. Densest translational lattice packing of non-convex polygons. *Computational Geometry: Theory and Applications*, 22:205–222, 2002.

[14] V.J. Milenkovic and K. Daniels. Translational polygon containment and minimal enclosure using mathematical programming. *International Transactions in Operational Research*, 6:525–554, 1999.

[15] Alexander P. Morgan. *Solving Polynomial Systems Using Continuation for Scientific and Engineering Problems*. Prentice-Hall, Englewood Cliffs, NJ, 1987.

[16] Bernard Mourrain, Jean-Pierre Técourt, and Monique Teillaud. Sweeping an arrangement of quadrics in 3d. In *Proceeding of the 19th European Workshop on Computational Geometry*, pages 31–34, 2003.

[17] Elmar Schömer and Nicola Wolpert. An exact and efficient approach for computing a cell in an arrangement of quadrics. *Computational Geometry: Theory and Application*, 2003. To appear in Special Issue on Robust Geometric Algorithms and their Implementations.

[18] Ron Wein. High-level filtering for arrangements of conic arcs. In Rolf H. Möhring and Rajeev Raman, editors, *Algorithms - Esa 2002, 10th Annual European Symposium*, volume 2461 of *Lecture Notes in Computer Science*, pages 884–895, Rome, Italy, 2002. Springer.

[19] Nicola Wolpert. Jacobi curves: computing the exact topology of arrangements of non-singular algebraic curves. In *Proceedings of the 11th ACM Symposium on Algorithms*, pages 532–543, 2003.

[20] Cheė Yap. Robust geometric computation. In J. E. Goodman and J. O'Rourke, editors, *Handbook of discrete and computational geometry*. CRC Press, Boca Raton, FL, second edition, 2004.